Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

APLICACIÓN MULTIPLATAFORMA PARA REPARTO DE NOTICIAS

Autor: Daniel Alonso Pardo
Tutor: Marcos Díez Guerra
Ponente: Simone Santini

Enero 2017

APLICACIÓN MULTIPLATAFORMA PARA REPARTO DE NOTICIAS

Autor: Daniel Alonso Pardo
Tutor: Marcos Díez Guerra
Ponente: Simone Santini

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero 2017

Resumen

Resumen

Al desarrollar una aplicación para ofrecerla al público general, es deseable poder ejecutarla desde una amplia variedad de plataformas, para tener el mayor mercado posible. Sin embargo, una mala elección de técnicas de desarrollo puede resultar en deuda técnica y sobrecostos innecesarios.

En esta tesis se explica el razonamiento detrás de la elección de tecnología para el desarrollo multiplataforma (como combinación de webapp y aplicación nativa, a través de Cordova), cómo afecta la workframe elegida (Meteor) al diseño de la arquitectura y los resultados de las decisiones tomadas.

Palabras Clave

Desarrollo multiplataforma, Android, iPhone, Meteor, Cordova, aplicaciones web, programación reactiva.

Abstract

When developing an application aimed at the general population, it is desirable for it to be usable from a variety of platforms, to have as many potential clients as possible. However, a poor choice of development techniques may result in technical debt and cost overruns.

In this thesis the reasoning behind the choice of cross-platform development technology (as a combination of webapp and native app through Cordova) is explained, as is the way in which the chosen workframe (Meteor) affects the design of arquitecture, and the results of these decisions.

Key words

Cross-platform development, Android, iPhone, Meteor, Cordova, webapps, reactive programming.

Agradecimientos

Esta tesis va dedicada a mi padre, que siempre ha intentado comprenderme, a mi madre, por estar siempre ahí, y a mi tía, por no dejar de alentarme.

Índice general

Índice de Figuras

1 Introducción

1.1. Acerca del problema multiplataforma

En la actualidad, existe una multitud de distintas plataformas como ordenadores con Windows o móviles con iOS. Esto supone un reto para cualquier empresa que quiera desarrollar una aplicación de software: por un lado, el mercado potencial de una aplicación es más amplio si se da soporte a más plataformas. Por otro, cada una aumenta de forma impredecible el costo de desarrollo de la aplicación.

Una plataforma puede diferir de otra de muchas maneras. Pueden usar sistemas operativos distintos, puede que librerías disponibles para uno no lo estén para otro, incluso es posible que el mismo lenguaje sea compilado de forma sutilmente distinta por distintos compiladores. Además, los componentes físicos de un sistema también son diferencias a salvar: desde pantallas más grandes o más pequeñas hasta periféricos completamente distintos, como los mandos de una consola frente a la pantalla táctil de un móvil.

Además, en la práctica estas diferencias no son las únicas que dificultan el desarrollo: mantener la coherencia visual y funcional en distintos entornos, gestionar los distintos bugfixes que pueden existir solamente en un entorno pero no en otro, y decisiones de alto nivel como extender el soporte a una plataforma completamente nueva son actividades que pueden resultar muy difíciles si no se han tenido en cuenta desde el principio.

Un proyecto multiplataforma debe mitigar todos estos costes eligiendo metodologías y herramientas adecuadas.

Este proyecto se centra en el desarrollo de una aplicación para prestar un servicio de distribución de noticias, con especial atención al aspecto multiplataforma:

- Plataformas para las que se ofrecerá soporte y por qué.
- Proceso de selección de tecnología y herramientas para facilitar el desarrollo.
- Diseño final de la aplicación y particulares de su implementación.

2

Estado del arte

Existen muchas aplicaciones diseñadas para funcionar en múltiples plataformas. Aquí se han señalado algunas de alcance similar a la de este proyecto.

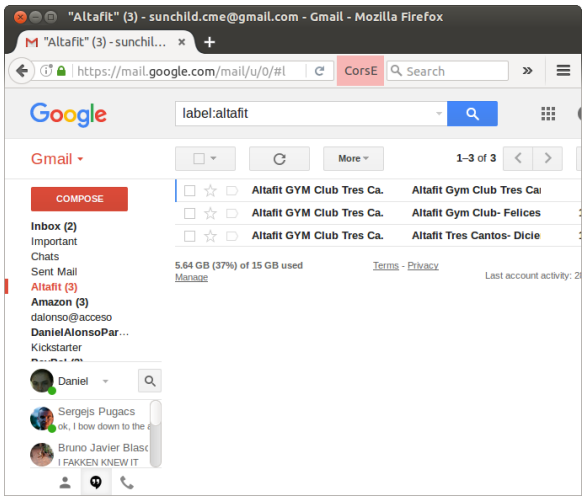

Figura 2.1: Gmail

Gmail (fig. 2.1) tiene, aparte de su propia aplicación nativa para plataforma móvil, una webapp a ser accedida desde cualquier browser. En este caso se resuelven las diferencias entre móvil y sobremesa detectando en qué tipo de plataforma está y sirviendo páginas distintas en uno y en otro. Mantiene la mayor parte de la funcionalidad, pero para mejorar la usabilidad cambia marcadamente la disposición y tamaño de elementos.

Google Maps (fig. 2.2) preserva mucho mejor la apariencia entre plataformas, sacrificando poca funcionalidad. Esto es porque no hay necesidad de mostrar muchos elementos, como es el caso en gmail donde hay varios tabs y muchos emails siempre a la vista.

Reddit ofrece una aplicación no basada en html (fig .2.3) para cargar de forma más eficiente sus contenidos, desarrollado aparte de la página web para browser. Sin embargo sigue siendo una aplicación multiplataforma al existir tanto versiones para Android como para iPhone.

De forma similar, la aplicación de Facebook (fig .2.4) existe aparte de la versión web, para

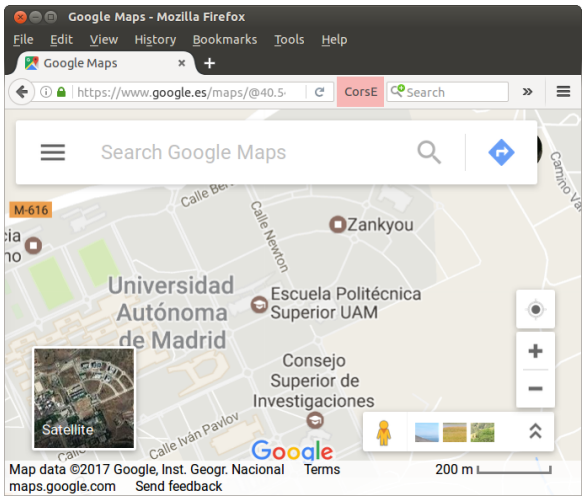

Figura 2.2: Google Maps

múltiples plataformas. A diferencia del resto de los ejemplos, la versión para browser está diseñada para adaptarse *peor* a las plataformas móviles, para fomentar el uso de la aplicación nativa en ellas.

Whatsapp (fig. 2.5) también tiene versiones como app nativa multiplataforma y como webapp, pero curiosamente empezó como app nativa y más tarde se creó la versión para web, justo al revés que en el resto de ejemplos.

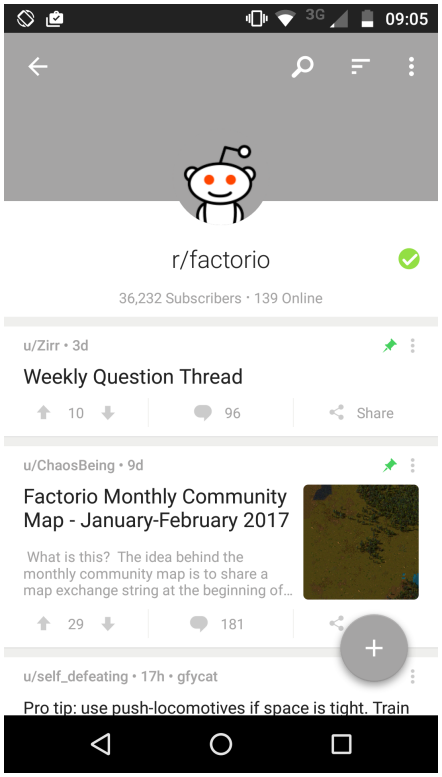

Figura 2.3: Aplicación móvil de reddit

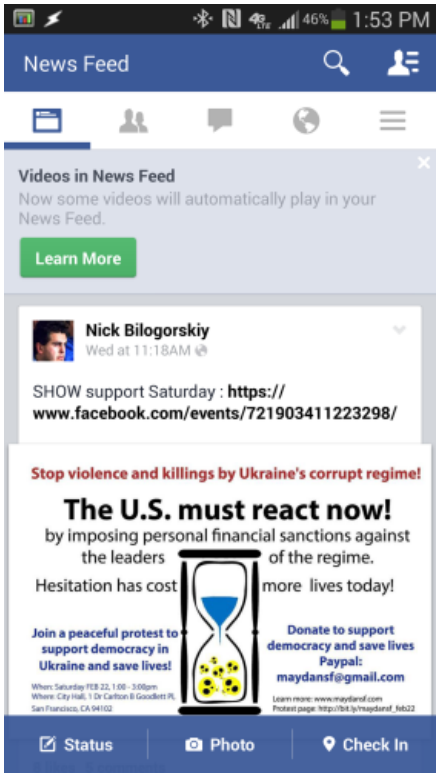

Figura 2.4: Aplicación de facebook

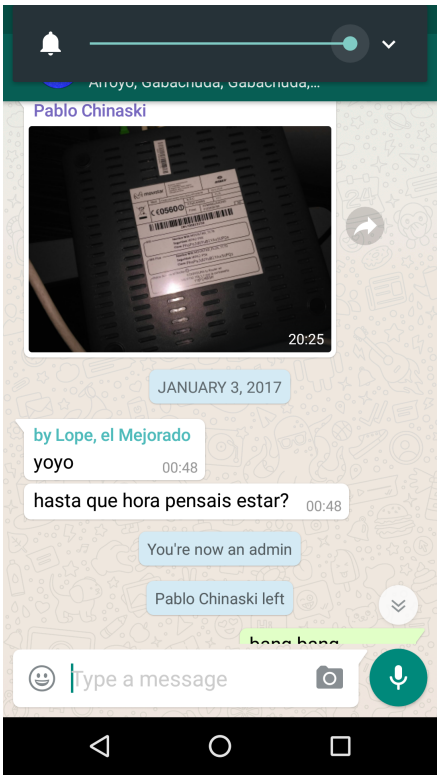

Figura 2.5: Whatsapp

3 Análisis

3.1. Objetivos del programa

Esta aplicación sustituirá a otra ya existente diseñada para para iOS. Servirá de apoyo a la página web de la empresa, en la que los usuarios pueden ver los canales de noticias a los que están suscritos y además realizar operaciones avanzadas sobre ellas o administrar sus cuentas. La aplicación a diseñar debe ser una versión simplificada, dirigida únicamente a supervisar canales y buscar noticias exclusivamente. Para ello deberá conectarse al servidor de noticias de la empresa a través de su API RESTful, que es el que realizará las tareas de búsqueda, recuperación de noticias y login del usuario.

Las noticias de un usuario se agrupan por "media", dependiendo del tipo (radio, periódico, vídeo o página web), "feed" que es una categoría definida por el mismo cliente, y "newsgroup", que es cualquier subcategoría dentro de ésta.

3.2. Definición de requisitos

3.2.1. Requisitos funcionales

- Permitir la autenticación del usuario a través de la base de datos de la empresa.
- Mostrar al usuario los distintos medios de comunicación que tiene disponible.
- Mostrar al usuario los canales y subcanales de noticias (feeds y newsgroups respectivamente) a los que está suscrito dentro de cada tipo de medio de comunicación.
- Mostrar al usuario las noticias de un canal concreto según la fecha y términos de búsqueda elegidos.
- Para cada noticia, mostrar su texto y sus metadatos.
- Para cada noticia, mostrar su fuente original. Esto puede ser un pdf, audio, vídeo o un link a una página web.

3.2.2. Requisitos no funcionales

- Proveer al usuario de una interfaz gráfica fácil de utilizar.
- Utilizar el servidor RESTful de la empresa para obtener credenciales y recuperar los datos de las noticias..
- Enmascarar los retrasos en la conexión con el servidor RESTful de la empresa.
- Desarrollar la aplicación para desplegarla en Android e iPhone.
- Minimizar los recursos necesarios para desplegar la aplicación.

4

Diseño e Implementación

4.1. Herramientas y tecnología

4.1.1. Selección de tecnología

Los principales métodos para generar ejecutables en distintas plataformas son los siguientes:

- Compilar el mismo código para cada plataforma con distintos compiladores. En Android no se ejecuta código nativo sino bytecode de java, dificultando el desarrollo en lenguajes que no sean Java. Las librerías usadas tienen que estar disponibles en las diferentes plataformas, así que es típico centrar el desarrollo alrededor de una librería multiplataforma en particular. Por ejemplo, Unity como librería gráfica.
- Crear el programa con lenguajes script. Para ejecutarlo haría falta un intérprete en el sistema objetivo.
- Usar Java o cualquier lenguaje que se compile a bytecode. Para ejecutarlo hace falta una máquina virtual de Java en el sistema objetivo.
- Crear el programa como webapp. De forma similar a los dos métodos anteriores, esto necesita un browser como Firefox o Chrome que pueda interpretar HTML y CSS para la parte visible y Javascript para la parte lógica. Además de en la web, también puede distribuirse como programa standalone. Frameworks como Cordova envuelven la webapp con un browser ligero para que se comporte como cualquier otra app.
- Usar un framework diseñado para enmascarar todo el proceso de compilar en distintas plataformas. Algunos frameworks como Qt y 8th usan su propio lenguaje para definir el comportamiento de la aplicación y ofrecen sus propias bibliotecas para garantizar el mismo comportamiento en diferentes builds, compilando el programa tanto en código binario como bytecode usando el mismo código.

Como se ha visto en la sección de análisiss, la aplicación a ser desarrollada se caracteriza por no realizar cálculos intensivos (el servidor se encarga de todas las operaciones complicadas), requerir una GUI relativamente simple que facilite su uso, necesitar mostrar media de distintos

tipos (vídeo, audio, documentos pdf y páginas web de noticias) y contactar con un servidor a través de una API RESTful para hacer operaciones simples (como obtener una lista de noticias).

De los métodos mencionados en la lista anterior, el que mejor favorece el proyecto es el modelo webapp: HTML, CSS y Javascript han sido diseñados precisamente para mostrar información visualmente de forma independiente de la plataforma, y ya existe multitud de herramientas para facilitar operaciones típicas como conectar dinámicamente con un servidor y mostrar audio/vídeo/pdfs.

4.1.2. Meteor

Meteor es la framework con la que se ha desarrollado esta aplicación. Fue elegida por las facilidades que da al programador, de las cuales las más atractivas para este proyecto son: programación reactiva en Javascript, despliegue sencillo de la aplicación y conversión automática a app de iPhone y Android.

El objetivo de la framework de Meteor es hacer transparente algunos aspectos del diseño de páginas web bien conocidos, por ejemplo extendiendo los mecanismos de programación reactiva al servidor (lo que significa que un cambio en el servidor automáticamente se refleja en la página web del usuario final, incluyendo cambios en la base de datos o del código fuente), o facilitando inmensamente la interacción entre los elementos de una página y el código javascript que define su comportamiento.

Permite desplegar una aplicación a varias plataformas distintas, de las cuales las más notables son como página web y como aplicación nativa de Android o de iPhone (usando Cordova como browser dedicado).

Meteor implementa la reactividad usando Blaze, que usa un sistema de plantillas donde cada una tiene una parte definida en HTML y otra en Javascript. Compila el proyecto en una aplicación web donde todas las plantillas definidas son renderizadas por su propio motor DOM, que es el que vuelve a cargar los elementos HTML cada vez que cambia una variable reactiva en ellas. El siguiente código es un ejemplo del uso de estas plantillas:

main.html

```
<body>
  {{> hola}}
</body>

<template name="hola">
  <button>Pincha aquí</button>
  <p>Has pinchado el botón {{contador}} veces.</p>
</template>
```

La parte visual de un Template se especifica con la sintaxis de Spacebars, que extiende HTML. Lo que se ve en este fichero es una página web cuyo cuerpo solo contiene un Template, "hola", definido justo debajo dentro de el tag <template name="hola">. Dentro de éste hay una referencia a un atributo de este Template: {{contador}}. El comportamiento de estos atributos se define en javascript. Gracias al entorno reactivo, cada vez que el valor de un atributo cambia, la webapp se actualiza y muestra el valor actual sin necesidad de que el desarrollador añada lógica de actualización.

main.js

```
Template.hello.onCreated(function helloOnCreated() {
```

```
  // counter starts at 0
  this.counter = new ReactiveVar(0);
});

Template.hello.helpers({
  counter() {
    return Template.instance().counter.get(); }
});

Template.hello.events({
  'click button'(event, instance) {
    // increment the counter when button is clicked

    instance.counter.set(instance.counter.get() + 1);}
});
```

Para completar la parte lógica, solo hace falta inicializar una variable reactiva cada vez que se crea el Template (dentro de "Template.hello.onCreated"), crear un método ayudante asociado al Template para poder acceder a dicha variable (dentro de "Template.hello.helpers") y añadir un evento asociado a cualquier evento "click" que provenga de un botón HTML, que aumente el valor del contador. Una vez más, es el entorno el que se ocupa de la lógica de actualización. Las variables de tipo ReactiveVar indican al Template cuándo cambian, y son las que causan que ésta se actualice.

4.1.3. Librerías adicionales

Para apoyar al desarrollo del proyecto se ha usado las siguientes librerías externas:

poetic:materialize-scss

Es una versión empaquetada para Meteor del framework de CSS Materialize. Permite crear páginas web con el mismo aspecto que un app de Android, siguiendo las pautas del Material Design de Google. Para ello proporciona recursos Javascript y Sass. Por usar Sass, tiene como requisito previo el paquete fourseven:scss, un intérprete de Sass.

Siguiendo Material Design se simplifica el diseño de la GUI, y usando este paquete se simplifica su desarrollo. Ofrece la mayoría de elementos usados en ese estilo de diseño, y en la mayoría de los casos solo es necesario modificarlos con una hoja de estilo Sass.

iron:router

iron:router es un paquete para Meteor que permite administrar fácilmente las urls y a qué páginas lleva, además de permitir comportamientos complejos. No es el enrutador oficial de Meteor, sin embargo simplifica ciertas operaciones como relacionar una url concreta con un Template y cumple su función dado que no es necesaria ninguna función avanzada de enrutamiento en este app.

amplify

Este paquete de Meteor es un envoltorio para la memoria local de HTML5, y en este proyecto se usa para extender la funcionalidad de las variables reactivas de Meteor, resultando en una

herramienta muy útil para webapps. Permite usar la misma información en dos sesiones distintas, mientras que sin ella se perderían al cerrar la aplicación (como app móvil) o al refrescar la página web (en browser).

PDF.js

PDF.js es un visor de pdf basado en browser completo y embebible en páginas web, mantenido por Mozilla. Este proyecto usa una versión ligeramente modificada del original para mostrar todos los ficheros de pdf.

4.1.4. API de la base de datos

El servidor de noticias de la empresa dispone de una API HTTP para acceder a todos los servicios usados en el proyecto, desde los vídeos de una noticia de televisión hasta una búsqueda de noticias.

Los métodos disponibles en esta API (con su nombre usado durante el desarrollo) se resumen a continuación:

- getDomain: Recuperar dominio del servidor de noticias al que realizar peticiones. Esto es necesario porque hay un servidor exclusivo para ciertos clientes. Con esta petición se envía el nombre de usuario al servidor principal, y éste responde con el servidor de noticias que corresponda al usuario.
- getToken: Parte del sistema de auntentificación. Se envía el usuario y contraseña, si es válido se recibe un token que se deberá usar en el resto de peticiones HTTP para validarlas como peticiones de un usuario legítimo.
- getFeedsAndGroups: Recupera una lista con los feeds (canales de noticias) a los que está suscrito el usuario, y para cada uno sus newsGroups (subcanales).
- getAvailableMediaTypes: Recuperar la lista de medios a los que el usuario tiene acceso (prensa, televisión, radio o web).
- getNewsByNewsGroup: Recuperar todas las noticias de un día dado pertenecientes a un newsgroup dado.
- getNewsSearchResults: Recuperar todas las noticias de un newsgroup que contengan las palabras indicadas.
- getCalendar: Recuperar los días del mes en los que haya noticias para un mes y un newsgroup dado.
- getPressCutout: Pdf de la noticia indicada (solo prensa)
- getPressThumbnail: Una versión thumbnail del pdf de la noticia (solo prensa)
- getPressPages: En el caso de una noticia con múltiples páginas, el pdf de la página indicada (solo prensa)
- getRadioAudio: El fichero de audio de la noticia (solo radio)
- getTvVideo: El vídeo de la noticia (solo televisión)
- getTvVideoThumbnail: Thumbnail del vídeo de la noticia (solo televisión)

4.2. Arquitectura

La estructura de la aplicación se divide simplemente en el módulo GUI, que contiene tanto la parte de diseño gráfica como su funcionamiento, el módulo de memoria local, que gestiona toda la memoria persistente de la aplicación, y el módulo API, que contiene métodos para comunicarse con el servidor de noticias y para interpretar las respuestas.

- GUI: La parte visual de la aplicación (más la definición de su comportamiento). Este módulo será implementado con las herramientas de Meteor para webapps, juntando el diseño visual (con HTML y SASS) con la programación de su lógica (en Javascript) a través de las plantillas de Blaze. El motor DOM se encarga de interpretarlos y presentar la página.

  Toda la información de estado (como preferencias de usuario, estado de login...) o de contenido (noticias, canales...) se obtiene del módulo de memoria local.

- memoria local: Este módulo proporciona a la GUI los datos que necesite para su correcto funcionamiento. Todas las variables usadas son reactivas, por lo que la GUI se actualiza automáticamente cada vez que cambian. Éstos se dividen en tres categorías:

  - login: Contiene el estado de autenticación del usuario y realiza la operación de login/logout.
  - settings: Guarda en la memoria local el canal/subcanal por defecto que se deba mostrar al iniciar la aplicación.
  - cache: Gestiona los datos recibidos del servidor de noticias y peticiones a la misma. Como su nombre indica es un búfer entre el servidor y el módulo GUI que permite a este último poder cargar una página rápidamente sin tener que esperar a que los datos se actualicen. Realiza las peticiones y actualiza los datos viejos automáticamente en cuanto recibe una respuesta del servidor.

- API: Todas las llamadas a la API se realizan a través de este módulo, que hace de interfaz entre el servidor de noticias y la aplicación, incluyendo la conversión de los objetos JSON recibidos a una estructura de datos utilizable por el resto de la aplicación.

(diagrama con los siguientes componentes:) móvil(Gui <—> caché <–> módulo comunicación <—)—servidor/bd(–>API)

Para facilitar el manejo del estado del usuario, la GUI hará uso de un enrutador que redirija el usuario a la página de login si no está autentificado o si su sesión caduca.

Cabe mencionar que aunque Meteor permite también montar un servidor para sus webapps, si no tiene código específico que ejecutar (como es el caso en este proyecto) su única función es servir la página a browsers. Como la aplicación solo será distribuiida como app nativa, no se usará en producción.

4.2.1. Diseño de la GUI

El objetivo principal de la aplicación de cara al usuario es facilitar la supervisión de noticias a las que esté suscrito, por lo que debe mostrar toda la información relevante:

- Medios: Indicar al usuario si las noticias mostradas son de prensa, internet, radio o televisión.

- Canales: Mostrar el nombre del grupo al que pertenece los subcanales. Un usuario puede tener varios canales, y usarlos para organizar subcanales de distintas maneras.

- Subcanales: Son listas de noticias relevantes para dicho subcanal. Se ven actualizados a diario por el servidor de noticias.

- Noticias: Los artículos y grabaciones en sí, además de sus metadatos.

Los objetivos de diseño para la GUI son:

- Adaptar la GUI al uso de smartphones, usando elementos apropiados para una pantalla táctil.

- Minimizar el tiempo que el usuario tiene que gastar en navegación reduciendo el número de páginas distintas usadas.

- Desde la vista principal, mostrar toda la información necesaria para mantener al usuario orientado (que de un vistazo se vea el medio, canal y subcanal del cual se está cogiendo noticias, sin tener que realizar ninguna acción especial).

- Tener acceso rápido a todas las funcionalidades de la aplicación, como cambiar de un subcanal a otro o ver los detalles de una noticia.

4.3. Implementación

4.3.1. Módulo GUI

Este módulo con dos plantillas base: una que muestra la página de login y permite la autentificación del usuario, y una que junta el resto de plantillas para formar la aplicación propiamente dicha.

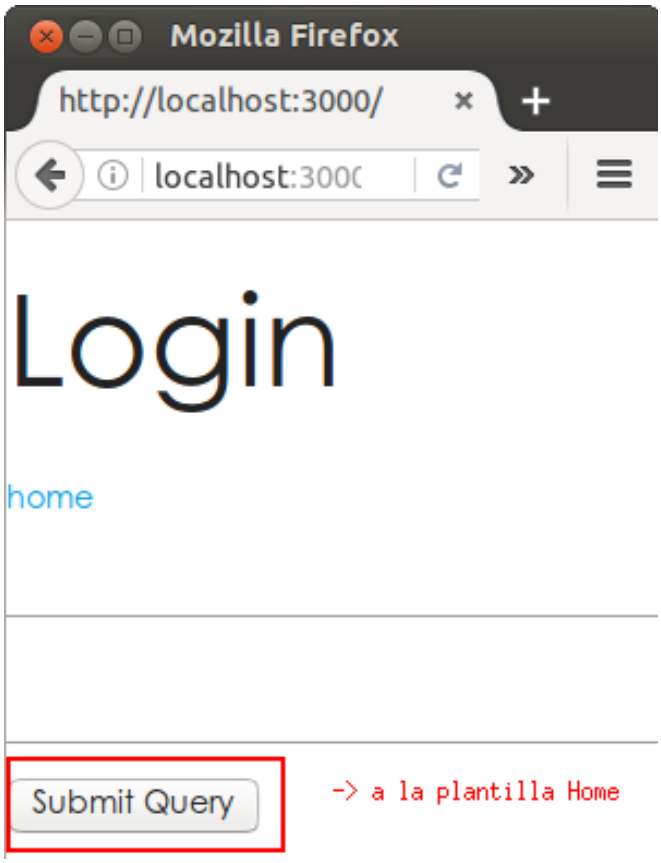

Figura 4.1: Página de login

La página de login realiza la acción de login a través del módulo de memoria local cuando se envía el formulario de usuario y contraseña. No realiza ninguna otra función.

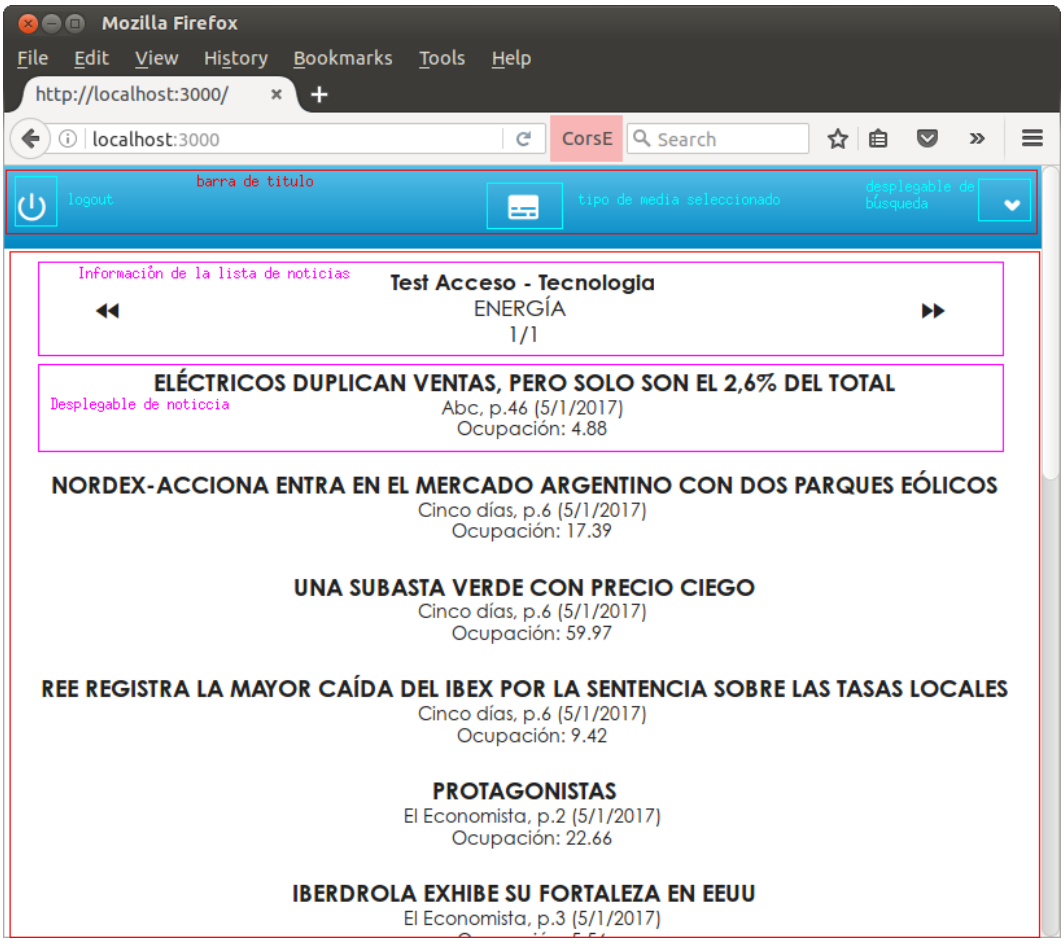

Figura 4.2: Página principal

Desde la página principal se ve una lista de titulares de noticias, encabezados por información sobre el canal, subcanal y página mostrada. Se muestra hasta quince titulares por página. Se ha decidido en contra de una lista infinitamente larga (estilo twitter, que carga nuevos tweets cuando llegas al final de la página) porque para realizar otras acciones como cambiar de canal habría que desplazarse otra vez hasta el principio.

En la barra azul que encabeza la página está un icono que representa el medio de las noticias (en este caso prensa), el botón de logout y el botón para desplegar la búsqueda/selector de canal. Se enseña al usuario qué iconos corresponden a qué medios en el desplegable de selección de canales (ver figura 4.4 y 4.5).

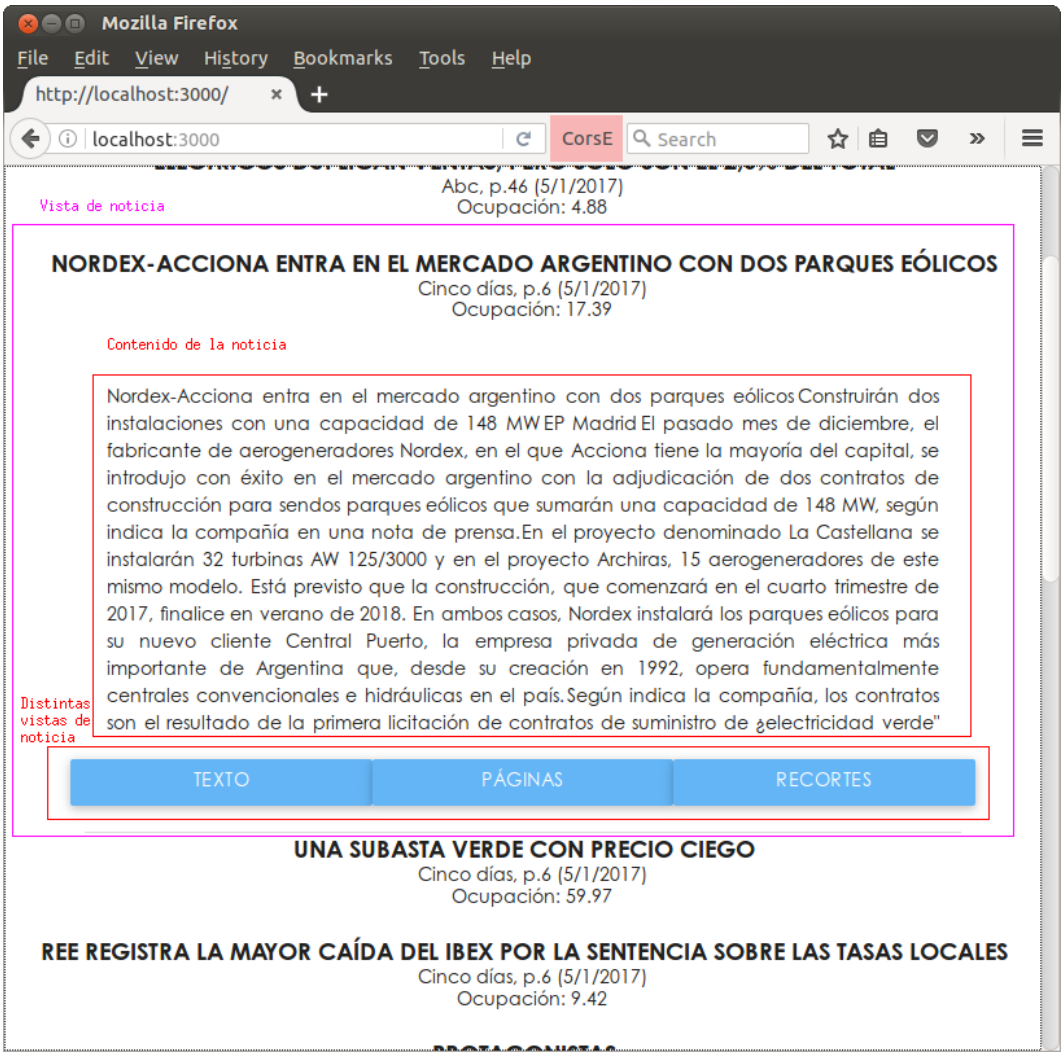

Figura 4.3: Noticia desplegada

Al seleccionar uno de los titulares, se despliega, revelando el contenido de la noticia, ya sea un vídeo, un fichero de audio o el texto del artículo (acotado si el texto es demasiado largo). Para las noticias que disponen de ello, hay botones que llevan a una vista detallada. Ver figura 4.7 para ver un ejemplo de la vista "PÁGINAS".

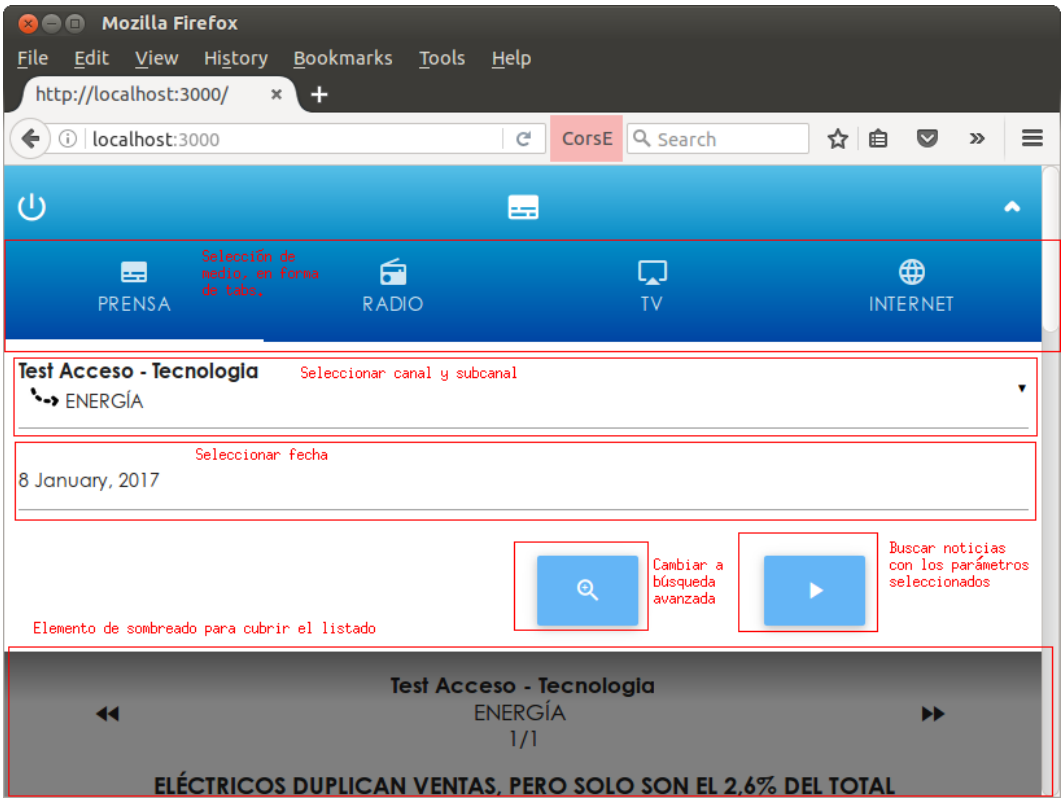

Figura 4.4: Desplegable de búsqueda

El desplegable deja al usuario elegir todos los parámetros que pueden definir a una lista de noticias (canal, subcanal y fecha). Los canales mostrados en el selector de canal/subcanal (ver figura 4.6) cambian al elegir otro medio. Al elegir fecha, se muestra en gris los días para los cuales no hay noticias.

Desde aquí se accede a la búsqueda avanzada.

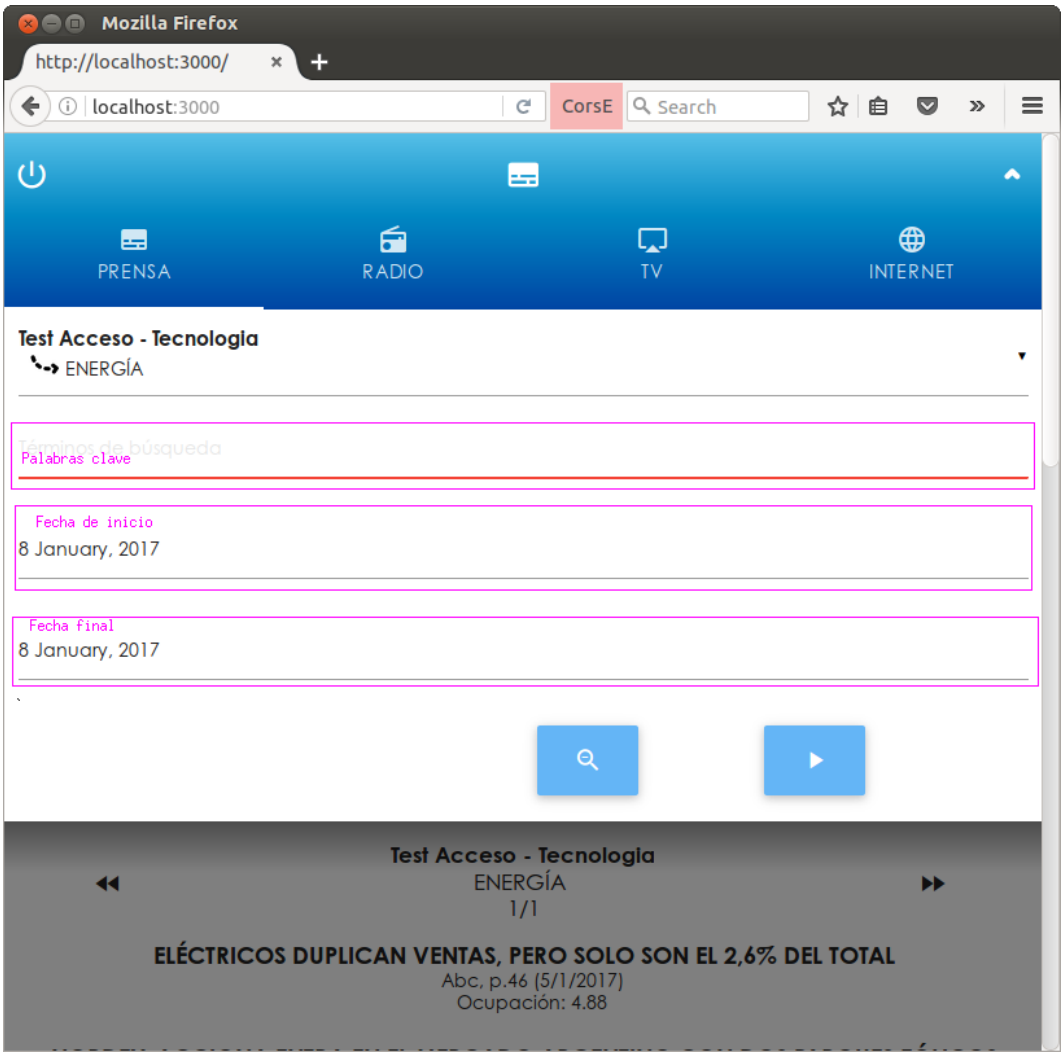

Figura 4.5: Búsqueda avanzada

Aunque a nivel de API las búsquedas avanzadas funcionan de manera muy diferente a la de simplemente pedir el subcanal para una fecha dada, se puede presentar como lo mismo pero con dos campos más, facilitando el adaptarse a él. En vez de pedir la fecha de un día, se admite un rango de fechas (opcional) y palabras clave (obligatorio).

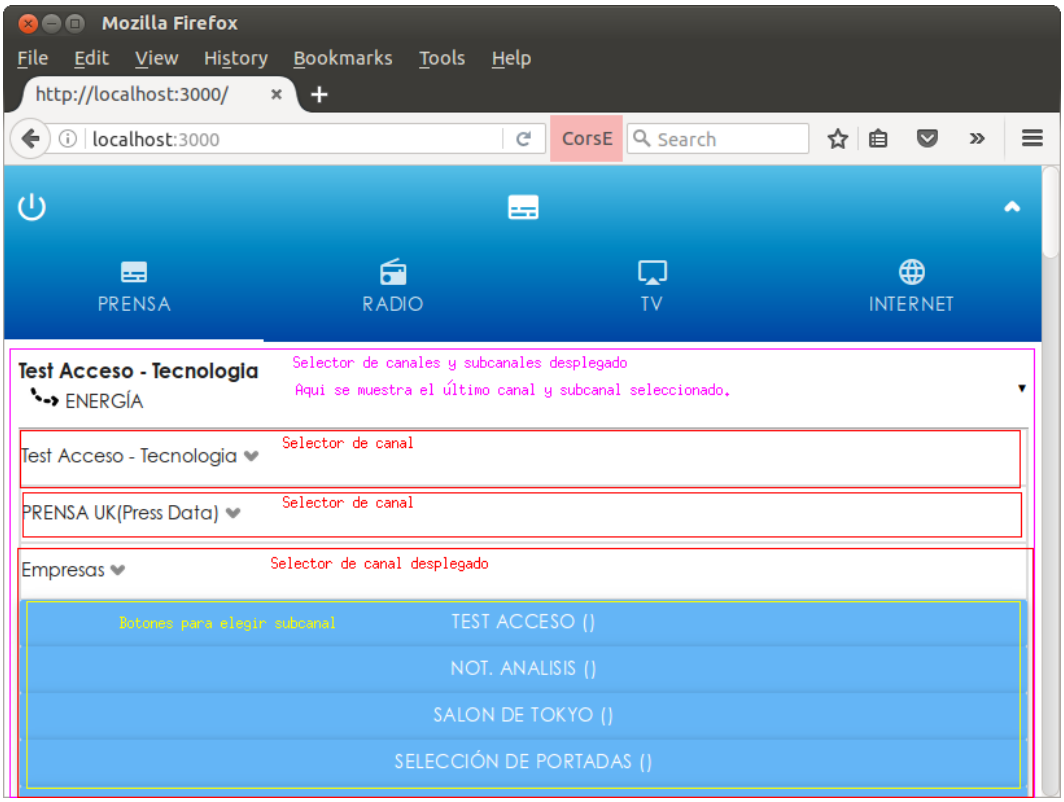

Figura 4.6: Desplegable de selección de canal y subcanal

Para seleccionar el canal/subcanal se utiliza desplegables anidados. En el primer nivel se muestra la lista de canales, y para cada canal se muestra todos los subcanales que pertenecen a ella. Al pinchar en uno de éstos, se queda grabado tanto el canal como el subcanal elegidos.

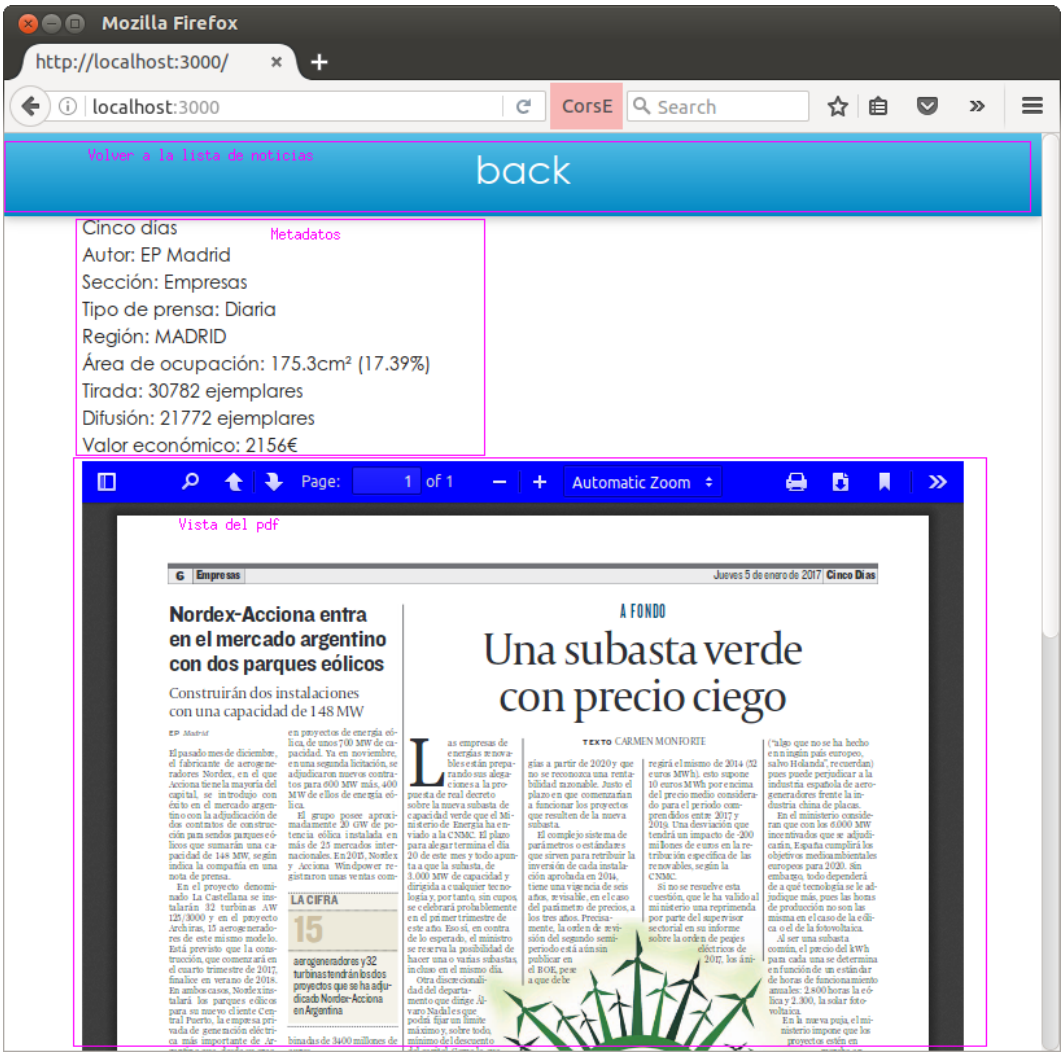

Figura 4.7: Vista detallada de noticia (página pdf)

En esta vista, accedida específicamente desde noticias de prensa a través del botón "PÁGINAS" (figura 4.3), se puede ver en detalle los metadatos de la noticia en cuestión y una versión pdf de la página original en la que se encuentra. Esta vista queda pendiente de mejoras de presentación. Otras vistas parecidas son la de recorte, similar en funcionamiento, y la de texto, que muestra el contenido del artículo entero sin acotar y está disponible también para noticias de internet.

4.3.2. Módulo de memoria local

Este módulo contiene todas las variables reactivas de las que depende el módulo GUI, tanto las que afectan su estado como el contenido que deba mostrar.

login

estructura del objeto login

```
login: {
  "login": function(usuario, contraseña, callback),
  "getLoginInfo": function(), //retorna {username, domain, token}
  "isLogged": function(),
  "logout": function()
}
```

El proceso de autentificación requiere dos pasos: primero, establecer el dominio a usar en las peticiones al servidor de noticias, y segundo, enviar el usuario y contraseña para obtener un token de seguridad, que se utiliza en el resto de peticiones para identificarlas como procedentes de un usuario registrado.

El módulo GUI utiliza las funciones de login (en la plantilla de login) y logout (en la plantilla de la barra de título).

settings

estructura del objeto settings

```
settings {
 defaultMedia: Object,
 defaultFeed: Object, //canal
 defaultNewsgroup: Object, //subcanal
}
```

Este módulo contiene información sobre el canal, subcanal y medio que se está viendo actualmente. El módulo GUI lo mantiene actualizado. Al usar variables locales que se mantienen de una sesión a otra, sirve para volver a abrir la última lista de noticias por defecto, al abrir la aplicación.

Se utiliza en las plantillas barra de título, selección de página y desplegable de selección de canales.

Cache

estructura del objeto cache

```
cache: {
  //cachea la lista de tipos de media disponibles al usuario
  mediaTypes: Object,
  //cachea la lista de canales y subcanales a las que está suscrito
  ////el usuario, por tipo de media
  feeds: Object,
  //cachea listas de noticias de subcanales, por fecha, subcanal y
  //// nž de página de resultados
  ngNews: Object,
  //cachea listas de noticias de búsquedas en subcanales, por
  ////términos de búsqueda, rango de fechas, subcanal y nž
```

```
  ////de página de resultados
  ngSearch: Object,
  //cachea información sobre las fechas en las que hay noticias,
  //// por mes y subcanal
  calendar: Object
}

cacheObject: {
  //objeto para la gestión de cada dato cacheado de su categoría
  //cada objeto miembro de caché es de este tipo

  //retorna un objeto reactivo con datos de la caché indicada
  get: function(cachedObjId),
  //indica si esta caché está pendiente de recibir
  ////una actualización del servidor de noticias
  isFetching: function(cachedObjId),
  //pide a la caché que actualice sus datos con el servior de
  ////noticias, si son más antiguas que secs segundos.
  update: function(cachedObjId, secs),
  //borrar los contenidos de esta caché
  erase: function(cachedObjId),
  //indica si el servidor ha respondido con un error
  hasError: function(cachedObjId)
}
```

El módulo caché pretende dar algo de transparencia a la hora de usar información procedente del servidor de noticias y separar su uso de su gestión.

En el módulo GUI cada plantilla que usa directamente cualquiera de las cachés también se encarga de su actualización a través de update().

Durante la actualización, la caché se marca como "fetching", evitando hacer la misma petición dos veces sin haber esperado a la respuesta.

4.3.3. Módulo API

estructura del módulo API

```
server: {
  url: {
    domain: function(),
    getPressCutout: function(newsId),
    getPressThumbnail: function(newsId),
    getPressPages: function(pageId),
    getRadioAudio: function(newsId),
    getTvVideo: function(newsId),
    getTvVideoThumbnail: function(newsId, width, height)
  },
  API: {
    getDomain: function(username, callback),
    getToken: function(username, password, callback),
    getFeedsAndGroups: function(mediaType, callback),
```

```
    getAvailableMediaTypes: function(callback),
    getNewsSearchResults: function(mediaType, options, callback),
    getCalendar: function(newsGroup, month, callback),
    getNewsByNewsGroup: function(newsGroup, options, callback)
  }
}
```

Este módulo es el que interactúa con el servidor de noticias. Compone las URLs para usarlas tal cual (en el caso de recursos como vídeos o pdfs), o a través del módulo mismo usando un callback para manejar los datos de la respuesta (para cuando la respuesta esperada es un mensaje JSON) de forma asíncrona.

Es usado por el módulo de datos para comunicarse con el servidor de noticias.

4.3.4. Manejo del login

El enrutador incluído a través del paquete iron:router se utiliza para implementar la detección de login. Normalmente se utiliza para asociar direcciones URL con plantillas. En esta aplicación, por ejemplo, relaciona la dirección raíz con la plantilla "homeSettings", y la dirección "/login" con la plantilla del mismo nombre.

En este caso, además, se hace uso de la capacidad del enrutador para cargar páginas de forma condicional. Se configura para detectar cuándo el usuario no está logueado. Si no lo está, carga la plantilla login en vez de cualquier otra. Además es reactivo, por lo que devuelve al usuario a la página de login con solo usar la función logout() del módulo login.

4.3.5. Pruebas

Las pruebas se realizaron tanto en un móvil con Android como en Chrome y Firefox en el ordenador de desarrollo. No se realizaron pruebas en iPhone al no disponer de uno durante el desarrollo. Las mayores diferencias entre browser y browser ocurren en el motor de renderizado. Firefox utiliza el motor Gecko, Safari usa WebKit, y Chrome usa Blink, y en plataformas móviles Cordova hace uso del browser nativo. Blink es un fork de un componente de WebKit, por lo que debería tener buena compatibilidad con ésta.

Para el ámbito del desarrollo de esta aplicación, se considera suficiente el uso de estas tres plataformas para las pruebas.

Uno de los obstáculos para el desarrollo de las pruebas es la necesidad de usar CORS (Cross Origin Resource Sharing) para acceder a cualquier recurso del servidor de noticias desde el código Javascript. Esto es por una medida de seguridad existente en los browsers que requiere que el servidor del recurso al que se está intentando acceder le de permisos a la página web que haya originado la petición. Hasta las fases finales del proyecto el servidor de noticias no estaba configurado para ello, por lo que solo se podía realizar las pruebas en un browser con una extensión para reducir la seguirdad del browser.

5 Conclusiones

5.1. Valoración del uso de aplicaciones web para el desarrollo multiplataforma

Durante el desarrollo de este proyecto se ha diseñado e implementado una aplicación web con el objetivo de desplegarlo como aplicación nativa en dos plataformas móviles, Android e iOs. Para ello se ha hecho uso de HTML, Javascript y SASS a través del workframe Meteor.

Los estándares de HTML y Javascript están bien soportados, por lo que los problemas del desarrollo multiplataforma se reducen a una de presentación, donde los elementos visuales deben funcionar para distintos tamaños y formas de pantalla. Por ello, cualquier webapp se puede usar desde casi cualquier plataforma de interés para una empresa que busca ofrecerlo a la población general.

Meteor es una framework que, una vez aprendida, permite desarrollar webapps de forma muy rápida. De sus características, las que más impacto han tenido son el sistema de plantillas y el entorno reactivo, por encapsular en cada plantilla su parte HTML con su lógica en Javascript. Además facilita las pruebas al hacer que la página web muestre automáticamente los cambios en el código fuente.

El desarrollo de este trabajo de fin de grado ha proporcionado al estudiante conocimientos sobre el desarrollo multiplataforma, programación reactiva, programación de aplicaciones web y diseño visual.

5.2. Trabajo futuro

El trabajo futuro pesa sobre dos áreas principales: implementar mejoras a la usabilidad, y mejorar el sistema de pruebas.

Acerca de la usabilidad, quedan pendientes mejoras de la UI, entre ellas la del estado del caché. Ésta enmascara el estado de la información. Mientras que evita tiempos de carga largos y esperas innecesarias cuando los datos ya cacheados coinciden con las de la base de datos, el usuario no sabe si la aplicación está a la espera del servidor o no. Se debe añadir un elemento a la UI que informe del estado.

En cuanto a las pruebas, se deberá realizarlas adicionalmente en iOs además de en móvil Android y browser, con emuladores apropiados o en vivo, para disponer de condiciones más reales.

Glosario de acrónimos

- **GUI**: Graphical User Interface
- **API**: Application Programming Interface

Bibliografía

[1] *Material design guidelines,* https://material.io/guidelines/

[2] *Meteor* https://www.meteor.com/

[3] *HTTP access control (CORS)*, https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS